High performance computing in fish school simulation by vortex sheet method

Fang Fang

May 20, 2015

1 Introduction of school simulation and vortex sheet method

Interactions of fish in a large school has been a mysterious topic for many years. People are interested in the reason why fish swim in school. Some theories believe that fish takes advantage of schooling as a way to reduce energy costs. The fluid drafting in fish school is not confirmed by mathematical simulation, and the hydrodynamics interaction is not well studied. In this report we will focus on a parallel algorithm of fish schooling simulation, using 2D vortex sheet method.

We model each fish by a single 1D rigid flapping wing, which is pitching with respect to the wing leading edge and wing orientation. The pitching motion is prescribed sinusoidally. The wing is free to move under hydrodynamic force and torque. As the wing moves in fluid, a vortex sheet is shed from wing trailing edge. The amount of circulation shed at each time-step is determined by the Kutta condition. Once shed, vortex sheet moves with flow with its circulation conserved.

Before we sketch the vortex sheet algorithm soon after, we recall the 2D Biot-Savart law. It expresses the complex fluid velocity conjugate $\overline{w(z,t)}$ at point z in terms of vorticity distribution γ in fluid,

$$\overline{w(z,t)} = \frac{1}{2\pi i} \sum_{k=1,\ldots,N} \int_{C_k^b + C_k^f} \frac{\gamma_k(s,t)}{z - \zeta_k(s,t)} ds. \tag{1.1}$$

where N is the number of wings, C_k^b and C_k^f denote the k^{th} wing and vortex sheet associated respectively, s is the arc length parameterization on curves. Biot-Savart law gives the method to calculate fluid velocity field by summing the contribution over all vortices. Once the fish school gets large, such summation can be extremely expensive. Now we state the simulation method briefly as follows. At each time step from $t_n \to t_{n+1}$, we update the positions of vortex sheet explicitly, by the second order Adam-Bashforth method.

$$\zeta^{n+1} = \zeta^n + \frac{3}{2}\triangle t w^n - \frac{1}{2}\triangle t w^{n-1}, .$$

Here ζ denotes the complex representation of vortex sheet point, and w is the complex fluid velocity at that point calculated by equation (1.1). After vortex sheet gets updated, the wing body is then solved iteratively through the quasi-Newton solvers. The constraint equations are provided by no-penetration boundary conditions of flow on wing bodies, together with balances of momentum and angular momentum,

$$\mathbf{u} \cdot \hat{\mathbf{n}} = \dot{\mathbf{x}} \cdot \hat{\mathbf{n}},\ Re\left(\overline{(w - \dot{\zeta})\hat{n}}\right) = 0 \text{ (complex form)},$$

$$m\ddot{\mathbf{X}}_k = F_k,\ I\ddot{\mathbf{p}}_k = T_k.$$

Therefore from time $t_n \to t_{n+1}$, the nonlinear equations need to solve is

$$Re\left(\overline{(w^{n+1} - \dot{\zeta}^{n+1})\hat{n}^{n+1}}\right) = 0$$

$$\begin{aligned}\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\triangle t} &= \frac{1}{2}\left(\dot{\mathbf{X}}^{n+1}+\dot{\mathbf{X}}^n\right),\\ m\frac{\dot{\mathbf{X}}^{n+1}-\dot{\mathbf{X}}^n}{\triangle t} &= \frac{1}{2}\left(\mathbf{F}^{n+1}+\mathbf{F}^n\right),\end{aligned}$$

$$\begin{aligned}\frac{\mathbf{p}^{n+1}-\mathbf{p}^n}{\triangle t} &= \frac{1}{2}\left(\dot{\mathbf{p}}^{n+1}+\dot{\mathbf{p}}^n\right),\\ I\frac{\dot{\mathbf{p}}^{n+1}-\dot{\mathbf{p}}^n}{\triangle t} &= \frac{1}{2}\left(T^{n+1}+T^n\right).\end{aligned}$$

The unknowns are the wing leading edge position as $\mathbf{X}^{n+1}$, wing orientation as $\mathbf{p}^{n+1}$, the vorticity distribution of wing γ^{n+1} and the circulation shed to vortex sheet Γ^{n+1}. We note that the fluid velocity w^{n+1} is calculated through equation (1.1) with the updated vortex sheets ζ^{n+1}.

2 Parallelism algorithm

As introduced earlier, at each time step the algorithm is separated into two big steps: first update free vortex sheets explicitly, then solve for the wing bodies iteratively by quasi-Newton method. For both steps, the most expensive parts lie in fluid velocity evaluation by equation (1.1), which is $O(n^2)$ expensive by direct summation, where n is the total number of discretization on wings and vortex sheets.

We consider a typical scale of fish school that contains $N = 10 \sim 1,000$ fish. Each fish body is represented by $M = 100$ degree of freedom, and the number of vortices on vortex sheet is growing at each time step, whereas for now we assume it is in a typical scale of $K = 1,000 \sim 10,000$. Therefore, the total body points can be as large as $NM = 10^3 \sim 10^5$, and the total free vortex discretization can be even larger as $NK = 10^4 \sim 10^7$.

We state our parallel algorithm in the two major steps respectively.

2.1 Parallelism of the explicit step

There are many parts of this step that can be paralleled. For now we focus on the most costly part – evaluating vortex sheet velocity using equation (1.1). It is a multiple summation over $N(M+K) \approx 10^4 \sim 10^7$ sources and $NK \approx 10^4 \sim 10^7$ targets, where fast multiple summation should be applied. We applied the 2D fast multiple method (FMM, [1]) with complex kernel $\frac{1}{z}$, after which we reduced the cost of velocity evaluation from $O(N^2K^2)$ to $O(NK\log(NK))$.

However, the velocity calculated using simple quadrature that summing over kernel $\frac{1}{z}$ is not accurate enough, where local corrections are necessary. There are two types of correction. Vortex sheet velocity induced by the vortex sheet itself requires the quadrature over smoothed kernel $\frac{\overline{z}}{|z|^2+\delta^2}$, which is not available by simple FMM algorithm. Another correction required is the accurate quadrature of vortex sheet velocity induced by wing bodies. When a vortex sheet gets close to a wing body, the singular kernel $\frac{1}{z}$ requires accurate quadrature to prevent big simulation error which looks like fluid penetration on the wing boundary.

We first state the parallelism of the first correction. We parallelize the self sheet correction over N vortex sheets, using shared memory OpenMp with C++ implementation. Each processor is assigned with a single vortex sheet correction, which is $O(K^2)$ complex by direct summation method. Self correction is local which makes no communication between processors. Each processor only gets the data of the vortex sheet it works on. After the computation is done, each processor writes its own data into the global output vector. By such algorithm we are able to reduce the serial $O(NK^2)$ computation time to $O(K^2)$ time over N processors.

To test this parallel self sheet correction algorithm, we generate N random vortex point sets from Matlab, as N fake sheets, each sets contain $K = 1000$ vortices. The fake test data preserves same flops needed as real fish simulation data. We distribute the $O(K^2) = O(10^6)$ flops over N processors. We test our code on Courant machines crunchy1, crunchy5 and crunchy6, which has 64 cores on each. In figure 2.1 we show the strong scaling and weak scaling on crunchy machines. For strong scaling test we set $N = 64$. There is no communication between processors. But as N becomes large, the data passing from Matlab to C++ can be slow.

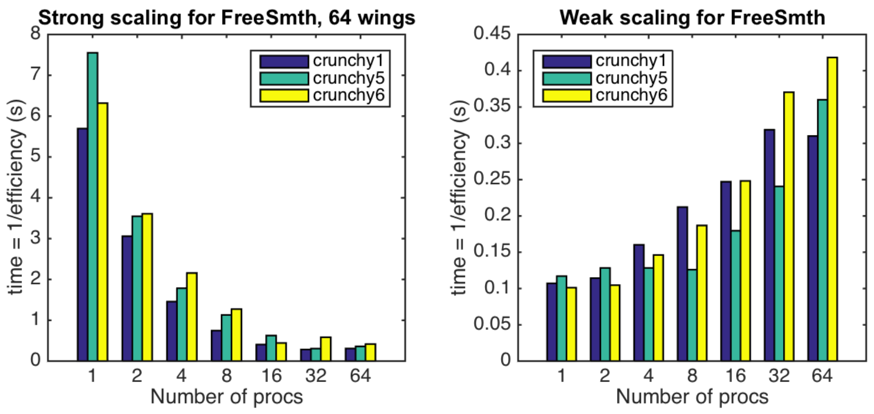

Figure 2.1: Strong scaling (left) and weak scaling (right) of parallel algorithm for vortex sheet self velocity correction. Tests are performed on Courant crunchy machines. For strong scaling tests $N = 64$ is fixed.

Besides parallelizing vortex sheet self correction, the second correction can be also parallelized. The second correction is to correct the velocity of vortex sheet induced by wing bodies. To replace the $\frac{1}{z}$ kernel, the accurate quadrature using a singularity subtraction technique is applied. For each vortex sheet, the correction needs to be applied over N wing bodies, which is of $O(NMK)$ expensive. So the serial cost of this step of correction is $O(N^2MK)$. We are working on parallel this step over N sheets and N bodies, which is expected to provide $O(MK)$ performance. This part of work is still in process.

2.2 Parallelism of the implicit step

At the implicit step, each wing is solved through the quasi-Newton solver independently, so that the solver applied on N wings can be distributed on N independent processors. This part os paralleling is in process.

Despite the the general parallelism of distributing N wing solvers on N processors, for a single wing solver, the computation is also expensive and can be parallelized. As in the explicit step, the main cost lies in the evaluation of wing velocity w^{n+1} through equation (1.1). This multiple summation is over $(N-1)M$ body source points and NK vortex sheet source points, with M wing target points. The fast multiple method with kernel $\frac{1}{z}$ can be applied for the wing velocity contribution induced by other $N-1$ wings, which reduces the direct summation cost of wing-wing interaction from $O(NM^2)$ to FMM complexity which is no more than $O(NM\log(NM))$.

The other part of velocity contribution is that induced by vortex sheets, which requires more accurate quadrature other than summation with kernel $\frac{1}{z}$. The quadrature we use is to approximate the circulation on vortex sheet by piecewise linear functions, and evaluate the integral of piecewise linear approximation analytically. For one single wing of one quasi-Newton iteration, the computation of wing velocity induced by all vortex sheets is of $O(NMK)$ complex. We distribute the velocity computation induced by N wings on N different processors, after which we then reduce the velocity contributions to one sum output. We expect the complexity reduces to $O(MK)$ with this parallelism.

We now test our parallel algorithm of summing velocity contributions with the accurate quadrature. As what we tested before, we generate N fake vortex sheets and $M = 900$ target points in domain $[-1,1]\times[-1,1]$, where each vortex sheet contains $K = 1000$ source points. We use shared memory OpenMp with c++ implementation again. The k^{th} processor gets information of the wing body and the k^{th} vortex sheet, then does the computation of accurate quadrature over the k^{th} vortex sheet, after which the velocity contributions from each processor get all reduced to the final sum of velocity. In figure 2.2 we show the strong scaling and weak scaling on crunchy machines. For strong scaling test we set $N = 64$. There is no communication between processors.

3 Conclusions

With above two parallelisms and FMM implemented in the current Matlab code, a speed up of five times has been observed for $N = 10$ flapping wings. We expect that after implementing all possible parallelisms discussed,

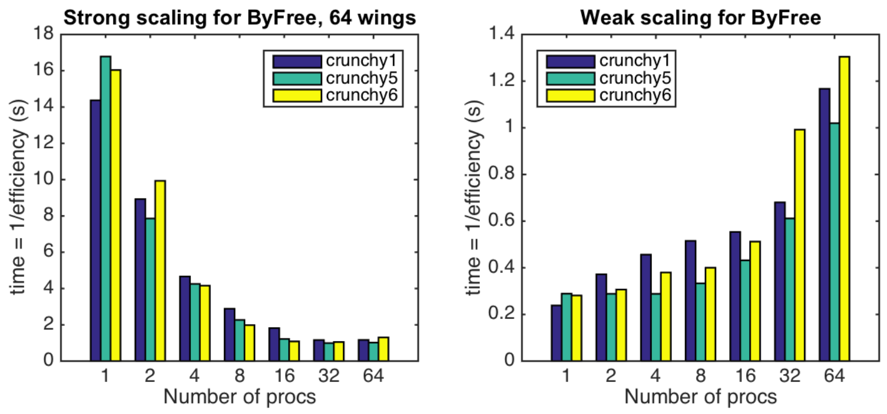

Figure 2.2: Strong scaling (left) and weak scaling (right) of parallel algorithm for evaluating body velocity induced by vortex sheets. Tests are performed on Courant crunchy machines. For strong scaling tests $N = 64$ is fixed.

the speed would not depend on the increasing number of fish. Also interactions between point vortices are complicated. A coarse grain model of point vortices might be helpful in computation.

References

[1] J. Carrier, L. Greengard, and V. Rokhlin, "A Fast Adaptive Multipole Algorithm for Particle Simulations," SIAM J. Sci. and Stat. Comput. **9**, pp. 669–686 (1988).